Object Oriented Programming and Design in Java

Session 20
Instructor: Bert Huang
Announcements

- Homework 4 due Monday, Apr. 19th (next class)
Review

- Homework tips
- Data Structures
  - Lists, Stacks, Queues
  - Sets, HashSet
  - Maps, HashMap
Today’s Plan

- Applications of queues, stacks, maps, sets
- Binary search trees
- Priority Queues (Heaps)
## Summary

<table>
<thead>
<tr>
<th></th>
<th>insert</th>
<th>insert at</th>
<th>remove</th>
<th>remove at</th>
<th>contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>lists</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>stacks/ queues</td>
<td>$O(1)$</td>
<td>X</td>
<td>$O(1)$</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>set</td>
<td>$O(1)$</td>
<td>X</td>
<td>$O(1)$</td>
<td>X</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>map</td>
<td>$O(1)$</td>
<td>$\sim O(1)$</td>
<td>$O(1)$</td>
<td>$\sim O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Data Type Applications

- Abstract Data Types allow well-organized design of data applications
- Design in terms of ADTs, most environments provide efficient implementations of standard ADTs
- Know which ADTs and data structures apply in different situations
Producer Consumer Queues

- Web server receives HTTP requests from browsers, puts request in queue
- Other threads remove from the queue, serve web pages to browsers
- Using a queue guarantees $O(1)$ operations and first-come-first-serve scheduling
Deques

• A deque is a queue and a stack
• Insert and remove from either head or tail
  • addFirst(e), addLast(e), getFirst(), getLast()
• ArrayDeque<E> implements Queue<E>
• LinkedList<E> implements Queue<E> and Deque<E>
Stacks for Method Calls

- When method is called, parameters and variables in its scope are pushed
- Once it is evaluated, it is popped
- Nested method calls populate a stack
  ```java
  System.out.println(scanner.next());
  ``
- Too many nested calls causes stack overflow, JVM out of memory
  ```java
  public void runForever() {
      runForever();
  }
  ```
Web Search by Word Sets

- Documents can be represented as sets of keywords
- Search for keywords by calling contains() on each document
- contains() and adding new document must be fast
- search $O(1)$ per document
- new document $O(k)$ for $k$ words

<table>
<thead>
<tr>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>fish</td>
</tr>
<tr>
<td>pet</td>
</tr>
<tr>
<td>fish</td>
</tr>
<tr>
<td>rice</td>
</tr>
<tr>
<td>chopsticks</td>
</tr>
<tr>
<td>chopsticks</td>
</tr>
<tr>
<td>deadlock</td>
</tr>
<tr>
<td>threads</td>
</tr>
</tbody>
</table>
Word Counting with Maps

- Natural extension to storing documents as word sets: word counts
- Each word maps to an integer count
  HashMap<String, Integer>
- Scan through document, increment count for each word
- “to be or not to be”
- $O(1)$ per word in document

<table>
<thead>
<tr>
<th></th>
<th>to</th>
<th>be</th>
<th>or</th>
<th>not</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>$+$</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Sorted Map ADT

- Subtype of Map (can get value by key)
- `SortedMap<K implements Comparable,V>`
- `SortedMap<K,V> subMap(K fromKey, K toKey)`
- `firstKey, lastKey, headMap, tailMap`
TreeMap

- Implements SortedMap
- put(), get(), contains() cost $O(\log N)$
- Uses an advanced binary search tree called Red-Black Tree
  - a balanced BST
- Slower than HashMap, but keys have order
Binary Search Tree

- Tree nodes have left and right children
- Left children are less than parent,
- Right children are greater than parent
- At each node, O(1) comparison determines which child to move to
- Depth of tree is the worst-case time for each operation
Due Dates with BST

- A calendar or to-do list program may store due dates in a BST
- Allows efficient search for date ranges
  - What’s due from today to Monday?
  - Show me things due after Monday
Priority Queue ADT

• Stores elements by priority (serves as the key)
  • Not really a queue, but used in similar applications
  • `add aka offer(E e)`
  • `deleteMin aka poll()`
  • `findMin aka peek()`
Heaps

- Binary tree with heap order property: keys of children greater than parent’s
- Running time:
  - $O(\log N)$ add,
  - $O(\log N)$ deleteMin,
  - $O(1)$ findMin
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>insert</th>
<th>findMin</th>
<th>get</th>
<th>get range</th>
</tr>
</thead>
<tbody>
<tr>
<td>lists</td>
<td>O(1)</td>
<td>O(N)</td>
<td>O(N)</td>
<td>X</td>
</tr>
<tr>
<td>hashmap</td>
<td>O(1)</td>
<td>O(N)</td>
<td>O(1)</td>
<td>X</td>
</tr>
<tr>
<td>BST</td>
<td>O(log N)</td>
<td>O(log N)</td>
<td>O(log N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>heap</td>
<td>O(log N)</td>
<td>O(1)</td>
<td>O(N)</td>
<td>X</td>
</tr>
</tbody>
</table>
Producer Consumer with Priority Queues

- Natural extension to using a simple queue, assign priority to all requests
- Consumer grabs the highest (lowest) priority element
- Is it worth the log N overhead? Depends on application
- If consuming is very fast, skip the fancy prioritization and just do it fast
Thread Safe Data Structures

- Since data structures are designed to be extremely fast, thread safety is omitted to avoid overhead

- Java has interface ConcurrentMap, implemented by ConcurrentHashMap

- and interface BlockingQueue, implemented by ArrayBlockingQueue, LinkedBlockingQueue
Threadsafe Wrappers

• Collections has static method
  \texttt{Collection synchronizedCollection(Collection c)}
  
• returns synchronized wrapper of \( c \)

• \texttt{synchronizedSet, List, Map, SortedMap}

• Returns \textit{decorated} object of anonymous class

• Each unsafe method is wrapped with an object lock
Reading

• http://java.sun.com/docs/books/tutorial/collections/implementations/index.html